Lecture 32

P, NP

Two Sides of Computation

Two Sides of Computation

Possibility

Two Sides of Computation

Possibility

Impossibility

Two Sides of Computation

Possibility: Algorithms

Impossibility

Two Sides of Computation

Possibility: Algorithms

Impossibility: Complexity Theory

Two Sides of Computation

Possibility: Algorithms

Deals with creating efficient algorithms for computational problems.

Impossibility: Complexity Theory

Two Sides of Computation

Possibility: Algorithms

Deals with creating efficient algorithms for computational problems.

Impossibility: Complexity Theory

Deals with proving non-existence of efficient algorithms for computational problems.

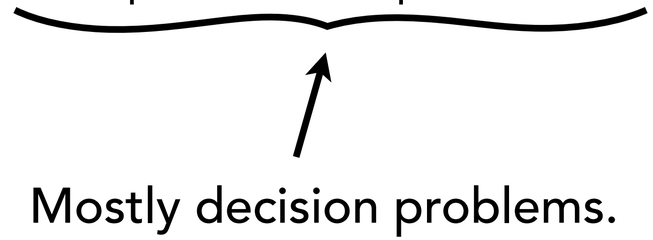
Two Sides of Computation

Possibility: Algorithms

Deals with creating efficient algorithms for computational problems.

Impossibility: Complexity Theory

Deals with proving non-existence of efficient algorithms for computational problems.



Search vs Decision Problem

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist.

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

SEARCH_PATH: Given a graph G and vertices $u,v \in G$, find a path from u to v or inform

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

> *SEARCH_PATH*: Given a graph G and vertices $u,v \in G$, find a path from u to v or inform if no such paths exist.

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

> *SEARCH_PATH*: Given a graph G and vertices $u,v \in G$, find a path from u to v or inform if no such paths exist.

Decision problems are computational problems where we have to decide whether

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

> *SEARCH_PATH*: Given a graph G and vertices $u,v \in G$, find a path from u to v or inform if no such paths exist.

Decision problems are computational problems where we have to decide whether a solution exists.

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

> *SEARCH_PATH*: Given a graph G and vertices $u,v \in G$, find a path from u to v or inform if no such paths exist.

Decision problems are computational problems where we have to decide whether a solution exists. For instance,

Search vs Decision Problem

Search problems are computational problems where we have to find a solution or inform that no solutions exist. For instance,

> *SEARCH_PATH*: Given a graph G and vertices $u,v \in G$, find a path from u to v or inform if no such paths exist.

Decision problems are computational problems where we have to decide whether a solution exists. For instance,

> *DEC_PATH*: Given a graph G and vertices $u,v \in G$, decide if a path from u to v exists.

Search vs Decision Problem

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation:

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *SEARCH_PATH* is polynomial-time solvable, then so is *DEC_PATH*.

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation:

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *DEC_PATH* is not polynomial-time solvable, then so is *SEARCH_PATH*.

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *DEC_PATH* is not polynomial-time solvable, then so is *SEARCH_PATH*.

We will focus on decision problems because:

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *DEC_PATH* is not polynomial-time solvable, then so is *SEARCH_PATH*.

We will focus on decision problems because:

- They are mathematically simple.

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *DEC_PATH* is not polynomial-time solvable, then so is *SEARCH_PATH*.

We will focus on decision problems because:

- They are mathematically simple.
- Lower bounds for decision problems implies lower bounds for search problems.

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *DEC_PATH* is not polynomial-time solvable, then so is *SEARCH_PATH*.

We will focus on decision problems because:

- They are mathematically simple.
- Lower bounds for decision problems implies lower bounds for search problems.

Search vs Decision Problem

Solving *DEC_PATH* via *SEARCH_PATH:*

1. Solve *SEARCH_PATH* on (G, u, v).
2. Answer **Yes** or **No** depending on the answer from the 1st step.

Observation: If *DEC_PATH* is not polynomial-time solvable, then so is *SEARCH_PATH*.

We will focus on decision problems because:

- They are mathematically simple.
- Lower bounds for decision problems implies lower bounds for search problems.

Note: We will assume that in the problems an input is always provided in encoded binary form.

Complexity Class P

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$,

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

- *MULT* = $\{\langle x, y, i \rangle \mid$ The ith bit of $x \cdot y$ is $1\}$

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of $x, y,$ and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of x, y, and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$

Decision problems can be expressed as sets.

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of $x, y,$ and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of $x, y,$ and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$

The polytime algorithm you have in your mind is actually an exponential time algorithm.

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of $x, y,$ and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of x, y, and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$
- *SORT* = $\{\langle x_1, x_2, \ldots, x_n, i\rangle \mid$ The ith bit of sorted sequence $(x_{k_1}, x_{k_2}, \ldots, x_{k_n})$ is $1\}$

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of x, y, and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$
- *SORT* = $\{\langle x_1, x_2, \ldots, x_n, i\rangle \mid$ The ith bit of sorted sequence $(x_{k_1}, x_{k_2}, \ldots, x_{k_n})$ is $1\}$

Some problems not known to be in **P**:

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of x, y, and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$
- *SORT* = $\{\langle x_1, x_2, \ldots, x_n, i\rangle \mid$ The ith bit of sorted sequence $(x_{k_1}, x_{k_2}, \ldots, x_{k_n})$ is $1\}$

Some problems not known to be in **P**:

- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$

Complexity Class P

Defn: **P** is the set of decision problems that can be solved in polynomial time, i.e., $O(n^c)$, where $c \geq 0$ and n is the length of the input.

Some problems in **P**:

Binary encoding of x, y, and i.

- *MULT* = $\{\langle x, y, i\rangle \mid$ The ith bit of $x \cdot y$ is $1\}$
- *PRIMES* = $\{\langle x\rangle \mid x$ is a prime number$\}$
- *SORT* = $\{\langle x_1, x_2, \ldots, x_n, i\rangle \mid$ The ith bit of sorted sequence $(x_{k_1}, x_{k_2}, \ldots, x_{k_n})$ is $1\}$

Some problems not known to be in **P**:

- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$
- *FACTOR* = $\{\langle x, l, u\rangle \mid x$ has a factor y such that $l \leq y \leq u\}$

Complexity Class P

Complexity Class P

Why P captures efficiency?

Complexity Class P

Why P captures efficiency?

- Time bounds less than n do not allow to even read the entire input.

Complexity Class P

Why P captures efficiency?

- Time bounds less than n do not allow to even read the entire input.
- In practice, we find algorithms with time n^2, n^3, etc. (not n^{30}, n^{100}, etc.).

Complexity Class P

Why P captures efficiency?

- Time bounds less than n do not allow to even read the entire input.
- In practice, we find algorithms with time n^2, n^3, etc. (not n^{30}, n^{100}, etc.).
- Problems with first polytime algorithm with high complexity, say $O(n^{20})$, usually get

Complexity Class P

Why P captures efficiency?

- Time bounds less than n do not allow to even read the entire input.
- In practice, we find algorithms with time n^2, n^3, etc. (not n^{30}, n^{100}, etc.).
- Problems with first polytime algorithm with high complexity, say $O(n^{20})$, usually get improved to, say $O(n^5)$.

Complexity Class NP

Complexity Class NP

Some problems not known to be in **P**:

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle \mid G$ has a hamiltonian cycle$\}$

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle$ | G has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle$ | G has a hamiltonian cycle$\}$
- *GI* = $\{\langle G_1, G_2\rangle$ | Graphs G_1 and G_2 are isomorphic$\}$

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle \mid G$ has a hamiltonian cycle$\}$
- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$

Not polytime solvable, but solutions are verifiable in polytime.

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle \mid G$ has a hamiltonian cycle$\}$
- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$

Not polytime solvable, but solutions are verifiable in polytime.

Informal Definition:

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle \mid G$ has a hamiltonian cycle$\}$
- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$

Not polytime solvable, but solutions are verifiable in polytime.

Informal Definition: NP captures the set of decisions problems whose solutions can be

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle \mid G$ has a hamiltonian cycle$\}$
- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$

Not polytime solvable, but solutions are verifiable in polytime.

Informal Definition: NP captures the set of decisions problems whose solutions can be verified in polynomial time.

Complexity Class NP

Some problems not known to be in **P**:

- *INDSET* = $\{\langle G, k\rangle \mid G$ has an independent set of size $k\}$
- *HAMILTONIANCYCLE* = $\{\langle G\rangle \mid G$ has a hamiltonian cycle$\}$
- *GI* = $\{\langle G_1, G_2\rangle \mid$ Graphs G_1 and G_2 are isomorphic$\}$

Not polytime solvable, but solutions are verifiable in polytime.

Informal Definition: **NP** captures the set of decisions problems whose solutions can be verified in polynomial time.

Note: **NP** stands for nondeterministic polynomial time.